The SyncMemo Project

Abraham Yaar {ayaar@mindspring.com}
David Starobinski {staro@bu.edu}
Ari Trachtenberg {trachten@bu.edu}

Department of Electrical and Computer Engineering
Boston University
Boston, MA 02215

Table of Contents

1.0 Introduction

The Pervasive Computing and Mobile Computing paradigms demand that a user have at least some access to his data over a wide range of computing devices. Periodic synchronization of data between varied devices is an enabling technology for these paradigms and an essential feature of any effective mobile and heterogeneous network architecture.

Network synchronization is often done naively, using wholesale data transfer to copy data from one location to the other and compare differences there. Clearly, this is suboptimal. Ideally, we would like a synchronization mechanism to transfer only the differences between two data sets and not the entire data sets themselves. The Characteristic Polynomial Synchronization algorithm (CPISync) allows us to achieve nearly this ideal bound by transferring data *proportional* to the number of differences. This algorithm can have important implications for a number of networking applications, including web-based caching and file replication. The idea to apply the CPISync algorithm to PDAs was first covered in the paper by Ari Trachtenberg, David Starobinski, and Sachin Agarwal, ``*Fast PDA Synchronization using Characteristic Polynomial Interpolation,"* in the Proceedings of *INFOCOM 2002*, pp.1510-1519, June 2002.

The SyncMemo project's goal has been to develop a prototype PC/PDA synchronization system to prove that CPISync and its variants can be implemented and provide superior performance to the standard PDA synchronization mechanism, slowsync. The SyncMemo project is thus basically a proof-of-concept for CPISync. This report is the guide to the entire SyncMemo project distribution set, which includes the (fully commented) source code for both the PC and PalmPilot applications, the test suites and helper software used to debug and test the system's performance, results from the performance tests, and presentation slides and graphics that we generated along the way.

1.1 Who Should Read This?

This report is mainly oriented towards the developer or tester who is interested in understanding the usage of the existing software in the project. This document treats the software at a high level, with minimal references to language specifics or to system hardware. A programmer interested in extending the code functionality will also find this report useful since it serves as a good overview - though not a substitute - for the source code. People who simply want to use the SyncMemo project files for their own personal data should be warned that this is not a user's manual (although it does cover the main aspects of using the system) and that the code base is not sufficiently debugged to use reliably on a daily basis.

2.0 Distribution Overview

2.1 Directory Structure

The following provides a comprehensive map of the directory tree of the main distribution, along with a short description of the software in each directory:

`root`

- `CConduit` - Contains the project directories for the PalmPilot Conduits that are written in C/C++
  - `SyncMemoCond` - The MSDEV project files for building the SyncMemo Conduit
- `CPalm` – Contains the project directories for all the PalmPilot software in C
  - `GenMemo` – The CodeWarrior project files along with the source code for the GenMemo application. Also stored here is a copy of the "random" data PalmPilot database, used to generate memos.
  - `SyncMemo` – The CodeWarrior project files along with the source code for the SyncMemo application.
- `PalmRoms` – Palm OS ROM files, used with POSE. Both *Debug* and *Release* versions of all 3.x ROMS are here. (*Note: As of this writing, newer ROMs are available online)
- `POSE` – All POSE files for v3.3. (*Note: As of this writing, there is a newer version of POSE available)
  - `Reporter_12` – Palm Reporter v1.2 files
- `Presentation` – Directories containing slides and graphics from project presentations.
  - `BU-UROP2001` – Presentation slides from the Boston University 2001 UROP presentation
  - `Sigcomm2002` – Presentation slides from the SIGCOMM2002 student poster presentation.
- `Source` – Primary directory containing the modular software components of the projects.
  - `Common` – Header files containing constants for all compiled project software.
  - `Palm` – Software modules (source and header files) used in software on the Palm platform.
  - `PC` – Software modules (source and header files) used in software on the PC platform.
- `TestTools` – All tools that automate processes in synchronization testing.
  - `Automated Testing` – VBscript files that entirely automate the PC-side of testing.
  - `ConsoleMemo` – MSDEV project directory for the PC automated memo generator.
  - `ConstChange` – MSDEV project directory for the PC automated constant modification program.

`Installer` – MSDEV project directory for the PC automated hotsync program installation tool.

`PalmAutoTest` – CodeWarrior project directory for the PalmPilot testing automation software.

`PCMemo` – MSDEV project directory for the PC-side memo entry program.

`WinNTL-5_0c` – A precompiled version of Victor Shoup's NTL library (source code and documentation are included).

2.2 Software Tools Description

The software tools in the project have been split into 4 categories, for ease of presentation. These categories and the applications that fit in them are not reflected in the directory structure of the distribution and are presented this way for ease of introduction!

2.2.1 Primary Applications

These 3 applications (and the software modules that they utilize) present the bulk of the work in the SyncMemo project. The applications allow the user to enter memos on the PalmPilot and PC, and implement the logic to synchronize between the two, within the PalmPilot's normal HotSync process.

2.2.1.1 The SyncMemo Application

This is the memo entry program (nearly identical to the built-in PalmPilot Memopad application) which is responsible for generating and maintaining the meta-data needed by the CPISync algorithm in response to user-input of memos. The UI of the program is divided into two windows: *memo edit* and *memo selection*. The *memo edit* window allows the user to enter text to create a new memo, or to edit the data of an existing memo. Once there are memos stored on the PalmPilot, the *memo selection* window lists the titles (first 25 characters) of all the memos present, alphabetically, allowing the user to choose one to edit. The memos themselves and their related meta-data that the SyncMemo application maintains are split into four databases: the SyncDocDB, the SyncIDocDB, the SyncHashDB and the SyncPolyDB. The contents of each of these databases are interfaced through their respective software modules, and are explained in those sections (see Section 3.1). For technical reasons, the SyncMemo application is also responsible for controlling the PalmPilot side of the conduit during a HotSync. This control is accomplished through use of the PalmCond module. Practically, this means that in order to synchronize using CPISync, the SyncMemo application must be installed on the PalmPilot so that its functions – when called by the PalmOS to synchronize – can call the proper parts of the PalmCond module. (NOTE: It is not sufficient to simply have SyncMemo installed, rather, the program must have been run at least once, in order for it to generate the SyncDocDB database, whose presence the Hotsync manager checks for, before the conduit is run.)

2.2.1.2 The SyncMemo Conduit

This application (which is really just a DLL called by the Palm HotSync Manager program during synchronization) is responsible for conducting the

whole synchronization process. The SyncMemo DLL is mostly just a wrapper that calls functions in the PCCond module to accomplish the actual synchronization. The DLL is responsible for initializing and collecting performance information, writing it to a file at the end of each synchronization round and putting a subset of that information in the HotSync Manager's Log (which is viewable by any Palm user). The detailed timing output information is written, by default, to a file called `Timeroutput.csv` located in the root directory of the C: drive. The output consists of values of the timer at specific locations in the conduit. These values are generally situated on both sides of an important function in the conduit, so timing information can be derived by subtracting two adjacent numbers. A sample `Timeroutput.csv` file is located in the root directory of the distribution. It contains the information about the points at which the timing numbers are collected. The timing output is designed to be plugged into a spreadsheet like the `results.xls` spreadsheet, also in the root directory of the distribution.

2.2.1.4 The PCMemo Application (Part I)

This application is the PC equivalent of BOTH the SyncMemo and GenMemo programs on the PalmPilot. In this section, we focus on the similarities to SyncMemo (where the user can generate their own memos) and leave the discussion of automatic memo generation to the PCMemo section under Testing Tools (section 2.2.2.2).

PCMemo allows the user to enter memos, just like in SyncMemo, with the titles displayed on the left pane of the window and the current memo displayed on the right. In other words, both the *memo edit* and *memo selection* modes from the SyncMemo application are displayed in one window. When starting the application, the user can type into the memo edit window directly to generate a memo, and when finished, can hit the "Save Memo" button to add the memo to the titles list. Memos that are in the titles list can be selected, and will appear in the memo edit pane. Once a memo is selected, the user must press "Save Memo" to clear the memo edit pane, if he wants to generate a new memo. In short, the operation of PCMemo is exactly analogous to the operation of SyncMemo, only everything is in one window. The one exception to this rule is saving. There is no save option in SyncMemo because memos are written automatically to the memo database whenever a user presses the "Done" button in the memo edit window, or whenever the PalmOS switches applications. However, in PCMemo, the user must explicitly save the memos that he has been working on by pressing the "Save File" button. Any number of memos can be generated and placed in the titles list, but unless the user presses the "Save File" button, the memos will NOT be written to file.

The file containing the saved memos is called `SyncMemo.dat` and PCMemo is given its location at compile time by the PALMPATH constant in the `constants.h` file. The SyncMemo.dat file is created within a subdirectory of the PALMPATH directory called `syncmemo` (this is also where the conduit checks for the PC's memos). The "Load File" button will also automatically load the `SyncMemo.dat` file from that location.

2.2.2 Testing Tools

When dealing with software systems that require complex data, it is useful to have programs that generate that data automatically, rather than having to do it one's

self. These programs are made to generate sets of memos easily and to test those memo sets for problems, before any synchronization is done.

2.2.2.1 The GenMemo Application

This application allows the developer/tester to create memos automatically, using "random" data from the `RandomDB.pdb` file (which MUST be installed on the PalmPilot that is running GenMemo). The data from the random database file never actually changes, but was generated using the C library's `rand()` function and converted to a Palm format database. The goal is to ensure that the automatically generated memos have the least chance of colliding in the hash space. The memos themselves will have the following format:

"MEMOXXX xxxx……."

where XXX represents the number of the particular memo in the generated set, starting at 0, (the inclusion of the memo number in the memo text allows for easier tracking of each individual memo when checking if synchronization has occurred properly) and xxxx represents the random data taken by a sequential scan through the `RandomDB.pdb` file. The parameters that the user can control are all displayed in the main UI window. The parameters are:

Number of memos to generate – (self explanatory)

Desired Memo size – This parameter controls the number of random characters that are included in each memo. IMPORTANT: There are only 50,000 characters in the random data file, so the product of the "number of memos" and the "desired memo size" should be less than or equal to 50,000, otherwise, the random data in the file will be exhausted.

Desired Number of Differences – This parameter allows for simple generation of symmetric differences between the Palm and PC memo sets. It will cause the GenMemo application to only create the number of memos specified *minus this parameter divided by two* (ie. #memos – (#diff / 2)). The same feature in GenMemo's counterpart on the PC (see 2.2.2.2) will cause the same number of memos to be generated, but they will be in the set {0…#memos-#diff, #memos-#diff / 2… #memos} so that the differences will be split evenly between the two. As a quick example, the memo numbers that will be generated on the PC and PalmPilot when the "Number of memos" parameter is 10 and the "Number of differences" parameter is 6 are:

Palm: {0,1,2,3,4,5,6}
PC: {0,1,2,3,7,8,9}

NOTE: Specifying a #differences greater than #memos will result in unpredictable behavior!

Once all the parameters are specified, simply pressing the button ('SyncMemo' for memos to be added to the SyncMemo databases and 'Standard Memo' for them to be added to the MemoPad database) will start the memo generation. The PalmPilot's cursor will freeze until it is finished generating memos (generally not longer than 10 min for even the largest number of memos, but always faster for Standard Memo than for SyncMemo). USAGE NOTE: When there are already existing memos in the destination database, the memo generation functions behave differently when generating memos for the SyncMemo database or for the MemoPad database. In the SyncMemo case, all existing memos are deleted and the new ones created in their place. In the MemoPad

case, the memo generation function will only generate memos of a number higher than the existing number of memos - so if there are 5 memos already existing and #memos is 10, only the last 5 memos will be generated (the reason for this is that after a memo deletion, even if the same memo text as the deleted memo is re-entered, it will be treated as a different memo during Hotsync, which makes testing MemoPad's performance a real pain).

2.2.2.2 The PCMemo Application (Part II)

The PCMemo application (written in C++ with the MFC document/view architecture) allows the developer or tester to create large sets of memos automatically, on the PC, by specifying a few parameters about the generated set. Although this application also allows the user to enter memos "by hand," this section will focus on the automatic memo generation features (for more on manual memo entry, see section 2.2.1.4)

The automatic memo generation features of PCMemo are accessed by pressing the "Fill In Data" button on the main window. This will pop up a dialog box with a list of parameters identical to those on the GenMemo application for the PalmPilot. In fact, PCMemo does on the PC everything that GenMemo does on the Palm, with a few exceptions: One, there is no way to generate memos directly into the Palm Desktop's MemoPad program (on the Palm, it is possible to generate memos for the MemoPad application). The reason for this is that Palm uses a proprietary file format when storing memos on the PC (whereas their Palm-side storage is simply text records inside a database). The second exception is that when using the #differences feature, PCMemo will obviously generate a complementary set of memos to those generated by GenMemo. In all other respects, PCMemo acts exactly like GenMemo. Provided that the file `random.dat` is in the same directory as the executable, PCMemo will generate the same "random" memos that would be generated in GenMemo. This is a very useful feature for testing synchronization with expanding data set size; you can create 10 memos on the Palm and 5 memos (using the same parameters) on the PC and expect there to only be 5 differences.

PCMemo also supports a useful debugging feature: hash collision detection. By compiling PCMemo with the constant `HASHDEBUG` defined in the `constants.h` file, PCMemo will output the hash values of all the memos it automatically generates into a file in the `C:\` directory called `hashout.tst`. This file is meant to be used with the `hashtester.xls` file to allow fast testing for hash collisions. For more on how to do this, see the `hastester.xls` section, 2.2.2.3.

PCMemo also has other constants in `constants.h` that affect its behavior. If the `JOKESFILE` constant is defined, then instead of using random data from the `random.dat` file to generate memos, PCMemo will use a list of jokes found in the file `jokes.txt` file. This feature is useful for presentation demonstrations! The final constant that PCMemo can use is the `EXPORT` constant. When this constant is defined, PCMemo will dump the actual texts of the memos in generates into a tab delimited text file in `C:\` called `memoout.txt`. This feature was used to insert memos easily into the Palm Desktop's MemoPad program since its import feature easily reads tab delimited files. However, this method of importing memos is not recommended, since it is much

simpler to simply generate memos on the PalmPilot using GenMemo and then hotsync with the PC (thus transferring the memos to the Palm's MemoPad program).

2.2.2.3 The `hashtester.xls` file

The `hashtester.xls` file is included in the distribution merely as a convenience to those developers who do not have background in spreadsheet programs, to save them from designing a program to do the same thing. What this spreadsheet file does is take, as input, the output of the PCMemo program under the constant `HASHDEBUG`, in the file `hashout.tst`. This file contains the hash values of each memo and the memo number that generated that hash. The values are separated by tabs, so that they can be easily copied from any text editor and pasted into Excel. `hashtester.xls` is structured to accept the `hashout.tst` file in the first two columns A and B. Once the data is in place, the user is supposed to sort columns A and B by ascending order in column B (NOTE: This is NOT the same as sorting just column B in ascending order; that is, after the sorting, the values of column A must remain in the same rows as the values they were next to in column B. This is accomplished by simply selecting both columns, and choosing the menu option **Data->Sort…** and selecting column B in the list box under "*Sort by*" [Excel 2000 & Excel 2002]). Each cell in column C displays the difference between the hash value in its row and the hash value in the row above it (thus, if the columns have been sorted correctly, this should always be a negative number, or zero). Cells in column D then check the cell in the same row in column C for a zero, and display FALSE if the don't find one, and TRUE if they do. A zero in column C (and a TRUE in column D) indicates a hash collision. The user can quickly scan through large memo sets by looking only at column D for the TRUE identifier. Once a hash collision is located, column A will identify the memo numbers of the colliding memos (the colliding memo numbers are in the cell at the same row at the TRUE and in the row above it). Since hash collisions cause unpredictable behavior in all the applications of section 2.2.1, it is recommended that any programmer/tester check the memo set generated by PCMemo, for a particular set of parameters, using hashtester, before synchronizing with that set.

2.2.3 Automated Testing Tools – PC

The SyncMemo project is sufficiently large and complex to require separate software to test it. In order to easily obtain large data sets, it is desirable that these test tools be easily automated. To meet this need, we have developed the following applications for preparing and executing test scenarios.

2.2.3.1 ConsoleMemo

The ConsoleMemo program is simply a command-line wrapper of the functionality offered by PCMemo. It has two modes, specified by an *action code* 0 or 1. Mode 0 is the functional equivalent of the "Fill In Data" mode in PCMemo. This mode lets the tester automatically generate memos with certain criteria. The next 3 command line arguments specify the number of memos to generate, the size of each memo and the number of symmetric differences between the PC and PalmPilot (for more on these, see PCMemo, section 2.2.2.2). There is no further input or output generated by the program, the memos are simply generated and stored in the `SyncMemo.dat` file in the

SyncMemo subdirectory of the PALMPATH directory. It should be noted that, like the "Fill In Data" mode of PCMemo, ConsoleMemo will delete all existing memos in the file when using this action code. Mode 1 allows the tester to append any single memo to the set of memos already in the SyncMemo.dat file. After the action code, any memo text can be entered as the next argument on the command line (although any memos containing whitespace must be in quotation marks). Again, no input or output besides the command line arguments is generated. The following is a summary of the command line options available, and their usage:

```
ConsoleMemo 0 <nummemos> <memosize> <numdiff>

ConsoleMemo 1 "Memo Text here"
```

The ConsoleMemo program allows for simple script automation of memo generation on the PC.

2.2.3.2 Installer

The Installer program allows the tester to specify files to be installed on the PalmPilot at the next Hotsync session from the command-line without using the Palm Desktop software. The program takes two command line arguments: the user name of Palm that the file should be installed on, and the full path to the file that should be installed:

```
Installer "User Name" "File Path"
```

The program itself simply takes the arguments and passes them to the *PltInstallFile* function of the Conduit SDK. Although it is true that any program can simply call this function by itself, the benefit of the Installer program is that a tester doesn't have to write a program to do this, he can simply write a script or bat file to take advantage of this feature.

2.2.3.3 ConstChange

The ConstChange program allows the tester to programmatically change the values for the constants in the constants.h file (or any other file that conforms to the same formatting standards). The definitions within the constants.h file follow a simple format: the "#define{SPACE}xxxxx{TAB}yyy{NEWLINE}" where 'xxxxx' is the name of the constant and 'yyy' is the integer value assigned to it (NOTE: ConstChange only works with integer constants). The program can modify the existing constant value in one of three ways:

1) Add to the constant a specified amount.
2) Subtract from the constant a specified amount.
3) Set the constant to a specified value.

The program's command line parameters are well documented in the comments at the top of the ConstChange.cpp file. This program is used within the AutoTest.vbs script to change certain constants before the PC-side programs are recompiled.

2.2.3.4 The AutoTest.vbs script

This Visual Basic Script file is the primary controller for automatic testing on the PC-side. Each experiment is composed of a variable number of rounds which are usually used to increment a constant over a certain range. Each experiment is defined by the following (in file) variables:

1) operationtype – this variable defines what other variables are changed between each round of the experiment (termed "operations" in the `AutoTest.vbs` file). Each operation is defined clearly in the *UpdateParameters* function within the source code file.

2) nummemos, memosize, memodifferences – these variables control the functions of the PCMemo program during each round, each according to its name. The values in the source code represent **starting** values, which are changed according to the operationtype constant at each round.

3) bound – This value acts exactly according to its name. When any of the variables in (2) reach or exceed this value (the value that is tested for depends on the operationtype), the operationtype is set to 0 which signals the end of the experiment.

This script is meant to run in conjunction (simultaneously) with its counterpart on the PalmPilot, called PalmAutoTest (described in section 2.2.4.1). AutoTest controls the PC-side of the experiment and PalmAutoTest controls the Palm-side of it. The experiments proceed in the following steps, in two phases, from the PC-side perspective:

Installation Phase

0) A timer, *x*, is reset and started.

1) The following programs are recompiled to update the binaries in the case that any of the constants in `constants.h` have changed:

a) The SyncMemo Conduit
b) The ConsoleMemo Program
c) The SyncMemo Application
d) The GenMemo Application

2) The PalmPilot binaries are set to install on the PalmPilot at the next hotsync.

3) The Hotsync manager is flushed of all conduits except the Install conduit

4) The script blocks for 60-*x* seconds, where *x* is the current value of the timer started in (0). This allows time for the PalmAutoTest application to initiate a hotsync on the PalmPilot and, in doing so, have the updated binaries installed on the Palm.

Synchronization Phase

5) The Hotsync manager is flushed of all conduits except the SyncMemo Conduit.

6) A timer, *y*, is reset and started

7) The PC generates a set of memos according to the current values of the variables nummemos, memosize and memodifferences.

8) The script blocks for 60-*y* seconds, where *y* is the current value of the timer started in (6). This allows time for the PalmAutoTest application to initiate a hotsync to synchronize the memos between the PC and Palm.

9) The variables in the script file are updated according to the operationtype variable. The next run begins at step 0, unless the operationtype has been set to 0 during this phase.

The experiments are designed in such a way that the only application that needs to be installed on the PalmPilot is the PalmAutoTest application. Since the PalmAutoTest function is designed to work with this file, it includes timings that work best if it is started 5 to 10 seconds before the AutoTest script is started. Also important to note is that both programs are timed in such a way as to allow 60 seconds for each of the phases. Because events are synchronous on the PalmPilot, if either of the phases takes longer than 60 seconds, the other phases are pushed back appropriately; however, events on the PC are asynchronous, so if a phase takes longer than 60 seconds, there could be major ramifications on the experimental results (for example: the Hotsync manager could be reset to the Install conduit before the synchronization hotsync takes place). The tester is encouraged to sanity-check the results of the experiments to guard against de-synchronization due to longer than 60 second phases. While the experiment is running, a popup box will appear detailing the current round and phase of the experiment. NOTE: The window will disappear by itself after the phase completes! Pressing the "OK" button will result in an early phase transition on the PC.

2.2.4 Automated Testing Tools – Palm

Unlike the PC, the PalmPilot has all of its automated testing functions contained in one application, so that the tester can easily prepare a PalmPilot for testing by installing only one program.

2.2.4.1 The PalmAutoTest Application

The PalmAutoTest application is the Palm-side counterpart to the AutoTest script on the PC (see section 2.2.3.4). It also executes in concert with the AutoTest script and has an identical variable set as that found in AutoTest, whose initial values can be set in the *initializeAutoTest* function.

The application uses PalmOS alarms to synchronize hotsync times (which must be initiated by PalmPilot software, as opposed to PC-side software) with the times that the AutoTest script is waiting for them. The experiments proceed in the following steps, from the Palm-perspective:

0) An initial delay is built into the program, so that the test process starts 20 seconds after the tester signals it to begin.

1) A timer, *x*, is set to expire in 45 seconds

2) A hotsync is initiated with the PC (this is when the updated binaries are installed on the Palm)

3) When timer *x* expires, a timer *y* is set to expire in 60 seconds.

4) PalmAutoTest calls Genmemo to create a new memo set according to the current values of the variables nummemos, memosize and differences. (Note: this can take a very long time with large values of nummemos!)

5) A hotsync is initiated with the PC (this is when the memo synchronization occurs)

6) The experiment variables are updated according to the operationtype variable. The next run begins at step (1) when timer *y* expires… if the operationtype variable has been set to zero, the next hotsync will still take place (it will install new binaries on the PalmPilot), but no hotsyncs after that will happen.

The tester should be warned that although there is no user interface indication that a test is in process, a timer may expire at any time, so there should be little or no use of the Palm while a test is running. If a timer expires while the Palm is in use (particularly if any of the memo applications are in use), the entire experiment may be halted with a fatal error message on the Palm. This is often due to the fact that PalmAutoTest tries to open the memo databases, which are opened by the memo applications themselves when they are running.

3.0 File/Database Formats

This section provides an overview of the purpose and format of all permanent storage files and databases generated by programs on the PC and the PalmPilot in the SyncMemo project. **The formats described in this section are illustrated in the slides of the BU-UROP2001 presentation (see Section 2.1: Directory Structure to locate them). The reader is strongly encouraged to reference those slides when reading this section.**

3.1 Palm Database Formats

In this section, the purpose of each database generated by the Palm-side applications in the SyncMemo project is explained and its format detailed. Unlike PC permanent storage, which uses the single abstraction of a *file* for a high-level storage unit, the PalmPilot has two such abstractions: the *database* and the *record*. The database is the highest level storage unit in the PalmPilot. Each database has a creator-ID and a unique name on the PalmPilot. A database can contain any number of records. Records are the closest thing on the PalmPilot to files on the PC – data can be read from/written to them at the byte level. Records are conceptually an array within a single database.

It is important to note that permanent storage on the PalmPilot is just battery-backed RAM, not a hard-disk. Because the PalmOS code itself resides in memory, there are strict access controls for both reading and writing databases and records in general. Because of these access controls, it is good to think of Palm storage as having the same performance as a PC hard-disk, rather than PC memory. For more information, the reader is encouraged to reference *The Palm OS Programming Bible.*

For each Palm database, there is a module that controls the implementation and exports the interface to the data stored in it. There is a "head" module that controls operations that involve more than one database called the PalmDb module – the remaining database modules are identified in the following sections.

3.1.1 The Alphabetized Document Database - SyncDocDB

The SyncDocDB is the first of two databases that contain the text of the memo entries from the SyncMemo application. Each record in this database contains a single memo entry in a standard character array format with a terminating null character at the end of the string (which is also the end of the record). The records are ordered within the database alphabetically so that the SyncMemo *memo selection* window can quickly and easily display an alphabetized list of memos to the user. The module responsible for this database is the PalmMemoDb module.

3.1.2 The Insertion Order Document Database - SyncIDocDB

The SyncIDocDB is the second of two databases that contain the test of the memo entries from the SyncMemo application. Each record in this database contains a single memo entry in a standard character array format with a terminating null character at the end of the string (which is also the end of the record). The records are ordered within the database according to their creation on the PalmPilot, with earlier memos occupying lower indexes than later memos. When memos are added to PalmPilot during data synchronization, they are added to the end of the SyncIDocDB database. This database is maintained so that the SyncHashDB, which holds the hash value of every memo, can have pointers from their hashes to the data that they are hashes of, without

having to reorder the pointers each time a new memo is inserted (see Section 3.1.3 for more information). The module responsible for this database is the PalmMemoDb module.

3.1.3 The Hash Database - SyncHashDB

The SyncHashDB is used to store the hash values of the memo entries from the SyncMemo application. There is only one record within this database that is an array of size NUMMEMOS (see the `constants.h` file) of `HashRecordType` structures (see the `PalmHashDb.h` file). Each structure contains a 16bit hash value and a 16bit integer which is the value of the index of the memo text that hashes to this hash value in the SyncIDocDb database. The database is organized as one record with a large array inside it, rather than a large array of small records, for performance reasons. During synchronization, the PalmPilot is presented with a list of hash values that the PC is missing, and must respond with the corresponding memo texts. The `HashRecordType` structure array inside the first record of the SyncHashDB database is sorted in ascending order to perform this hash-lookup quickly. Since the array is pre-allocated with size NUMMEMOS, in order to determine the number of valid hashes in the array, a program must check the 0^{th} index of the array. This index is always created with the SyncHashDB database (it holds the minimum hash value in the `hashvalue` field of the `HashRecordType` structure so that it remains at the 0^{th} index) and its `docindex` field of the `HashRecordType` structure contains the number of valid hashes stored in the array. The module responsible for this database is the PalmHashDb module.

3.1.3.1 Alternative Hash Database – SyncHashDB

There is an alternative implementation of the hash database on the PalmPilot that is structured identically to the first hash database, except that instead of the first record containing an array of `HashRecordType` structures, the first record contains an AVL tree of `HashRecordType` structures. The motivation behind this change was the hypothesis that insertion into the original hash array was time consuming, because the entire array after the insertion point would have to be moved forward by one index to make space for the insertion. With the AVL tree, the array is still there; however, all additions to the tree are placed at the end of the array with the only cost being modified pointers within the array. Unfortunately, performance experiments showed that although the AVLtree implementation was faster than the array implementation, this only showed when large numbers of memos where involved, and even then the time improvement was only minimal in the overall synchronization time. The module that implements this version of the hash database has an identical interface to the PalmHashDb module, and is thus interchangeable with it. It is called PalmHashDb2.

3.1.4 The Characteristic Polynomial Database – SyncPolyDb

The SyncPolyDB is where the SyncMemo application stores the characteristic polynomial evaluations for synchronization with the PC. The SyncPolyDB has the most complicated structure of the four databases on the PalmPilot. The database itself holds a varying number of records. The first record in the database is of size DIFFERENCES (see the `constants.h` file). The record is structured as an array of 16bit unsigned integers. The 0^{th} index of the array contains the number of memos stored

in the SyncDocDB database (the memo count is necessary for the CPISync algorithm). All remaining values in the array are the evaluations of the characteristic polynomial at specific points generated by the hashes of the memos stored on the PalmPilot. The index *n*, in the array of any specific evaluation is directly related to the *x*-value at which the characteristic polynomial was sampled, by the following equation:

$n = -(x+1)/2$ for x even
$n = x/2$ for x odd

When engaging in a standard CPI synchronization, the first record of the SyncPolyDB is downloaded in its entirety. However, during PROBSync synchronization, not all of the characteristic polynomials should be downloaded at once. The remaining records of the SyncPolyDB address this concern – they are identical to the first record of the database, only they have been separated into sections such that all of the remaining records concatenated are identical to the first one. Specifically, the second record in the SyncPolyDB database (the first of the "fragmented" ones) is identical to the first record (including the document count in the 0^{th} index) up until the PROB1 (see the `constants.h` file) index, which is the last index of the second record. The next record contains the first record's indexes PROB1 to PROB1*2 in its first PROB1 indexes. This continues, with each successive record twice as large as its predecessor, until a record reaches the size of DIFFERENCES. During a PROBSync, the records are downloaded one at a time, until there are enough evaluations on the PC to allow synchronization. The module that controls this database is the PalmPolyDb module.

3.2 PC File Formats

The permanent storage structure on the PC is much simpler than that on the PalmPilot; mostly due to the fact that the PC is not the performance bottleneck between the two and so it can afford to have a simpler, yet less efficient storage implementation. In fact, the PC has only a single file that contains all the information needed by the SyncMemo project applications, called SyncMemo.dat. All the applications interact with this file in the same way: they load the entire file into a data structure in memory, make changes to it and then re-write the entire file with the changes from memory. If synchronization between PCs rather than between a PC and PalmPilot were required, then this implementation may need to be changed.

3.2.1 SyncMemo.dat

The SyncMemo.dat file is located within the `syncmemo` subdirectory of each user's `Palm` directory. This location was chosen because it conforms to the built-in Palm applications, which each have subdirectories within the main `Palm` directory with ".dat" files in them.

The SyncMemo.dat file contains the memo text and hash values for the memo entries on the PC. The PC automatically generates the characteristic polynomials based on the hash values on-the-fly during synchronization, so they are not included in the SyncMemo.dat file. The SyncMemo.dat format is as follows:

Number of memos stored in the file : 2 bytes

Memo 1 Hash Value	:	2 bytes
Memo 1 Length (in characters)	:	2 bytes
Memo 1 Text Location In File	:	4 bytes
Memo 2 Hash Value	:	2 bytes
Memo 2 Length (in characters)	:	2 bytes
Memo 2 Text Location In File	:	4 bytes
…		
…		
…		
Memo X Text Location In File	:	4 bytes
Memo 1 Text (with null terminator)	:	(var)
Memo 2 Text (with null terminator)	:	(var)
…		
…		
…		
Memo X Text (with null terminator)	:	(var)

This file is read in from disk in a single pass (with seeks back and forth, but no data read twice) and place in an array of `memo` structures, with each structure containing the memo text and hash data. The module used to load and save data in this file is the PCMemoFile module.

4.0 Miscellaneous Comments

This section consists of a number of short subsections that cover some practical issues that do not fit well, or unduly complicate, any of the other sections.

4.1 Using POSE Effectively

The Palm OS Emulator (POSE) can be a very useful testing and debugging tool for a Palm programmer. Essentially, POSE can run any Palm-ready executable, and it can do so using a debug version of the PalmOS ROM which will halt the application on certain errors like unclosed records in databases and memory leaks that the release version of the PalmOS on an actual handheld will not report. The entire PalmPilot state can be saved to a file and reloaded so that reproducible bugs can be isolated quickly. Furthermore, debugging in CodeWarrior is simple and straightforward using POSE. Code is automatically uploaded when the debugger is started. POSE will halt and switch to CodeWarrior when any breakpoint in the code is encountered, and the user can step through program code and peek into memory as desired. POSE also includes a code profiler that can give a developer a good idea of which functions an actual PalmPilot will spend most of its time in.

POSE can also be set-up to hotsync with the PC as though it was a standard PalmPilot. Using PalmOS v3.3 requires some special, yet simple, tweaking to get this to work. POSE uses the PalmPilot's ability to hotsync over an Internet connection to hotsync with the PC on which it is running, using the loopback adapter IP address 127.0.0.1. Unfortunately, the PalmOS v3.3 ROMs don't include the software to do this natively, so the developer must install the file `NetSync.PRC` onto the Palm and must configure the following:

1) Under the Hotsync application, Menu Option…
    a) Modem Sync Prefs… select Network
    b) LanSync Prefs… select LanSync
    c) Primary PC Setup… enter 127.0.0.1 in the Primary PC Name field.
2) Confirm that under the POSE application menus (accessed by right-clicking anywhere in the POSE window), Settings->Properties, that the "Redirect NetLib calls to host TCP/IP" checkbox is checked.

It should be noted that neither the Palm timing, nor the communication speed of a hotsync approximate the speed of a real PalmPilot. Depending on the machine that POSE is run on, its performance can range from either greatly superior or greatly inferior to a real PalmPilot.

4.2 The Conduit Development Kit (CDK)

The Conduit Development Kit (CDK) provides a number of indispensable applications for testing conduit code. There are two primary applications that are of most use:

1) CondCfg.exe – This program allows the user to add conduits to the system. Conduit information can be entered and viewed through this application. Conduits can also be deleted or have their information change through this program. Overall, it is most useful simply to determine what conduits are installed on a particular PC.

2) CondSwitch.exe – This program allows the user to load and save entire conduit "profiles." This means that a user can save the configuration of all the conduits on a particular PC and load another configuration, all with this program. In the root directory of this distribution are three example profile files: `InstallOnly.txt`, which contains a profile with only the Install conduit enabled, SyncMemo.txt, which contains a profile with only the SyncMemo conduit enabled and StandardMemo.txt, which contains a profile with only the Standard PalmPilot memo program's conduit enabled. It should be noted that the locations of the conduits can change from PC to PC, since they are located in the `Palm` directory. Fortunately, the profile files are in standard text form and are simple to edit.

It is important to remember to restart the Hotsync manager after any modifications to the conduits on the PC.

4.3 Demonstration Installations

Moving the application files between PCs is not a simple process. Some of the binaries require non-obvious files to be in fixed locations that are not within the distribution tree (the most frequent external locations are the `Palm` directory or a user's directory within the `Palm` directory). These problems are somewhat alleviated by the PALMPATH and ROOTPATH constants found in the `constants.h` file. However, moving Microsoft Development Studio projects from one machine to another is also tricky (library and include directories seem to be stored in each machine NOT in each project file), so the most effective way of moving files is to recompile on a working machine but to change the PALMPATH and ROOTPATH constants to match the target machine.

5.0 Concluding Remarks

The purpose of the SyncMemo project was to demonstrate that the CPISync algorithm was practical and implementable. In doing this, we have developed an entire memo application on two different hardware platforms, we have implemented synchronization logic that conforms to the standard PalmPilot conduit interfaces, as well as constructed an entire test suite to assist the programmer in developing, debugging and testing the code. However, there remains plenty of work to do, both in maintaining, debugging and tweaking the performance of the existing code base, generalizing the existing code base and even implementing new synchronization algorithms to work within the framework described here. This section covers some of these issues and directions in greater detail.

5.1 Code Maintenance

The code for this project was developed over the course of two years, in 4 different languages and across two hardware platforms. Unfortunately, because of this and the experimental nature of the work, it was impossible to determine a good framework for the code before it was written. Rather, the current code's loose organization is something that was pulled together from the point at which we believed we had coded all the needed functionality. Though there is some basic modularity (namely, the 11 code modules) in the existing code and a hierarchy between those modules, the organization is not enforced, nor is the code designed to enforce it. Constants are global and hard-coded at each compilation. There are cases of overlapping functionality implemented in different modules. Overall, although this document goes some of the way toward addressing the confusion that these issues raise, a better solution would include the following:

1) Reimplementation or incorporation of the existing code into C++, where modules can be better constructed as objects. A single language of implementation would also help.

2) A comprehensive assessment of the constraints on parameters. For example, the `RandomDB.pdb` file contains only 50,000 characters in it, so when generating random memos, the product of the number of memos and the memo size needs to be less than 50,000. These constraints need to be documented and should also be enforced in the code.

3) Finally, a comprehensive diagram of the modules (along the lines of the system diagram in the BU-UROP2001 presentation directory) should be generated, detailing their interfaces (function calls and variables) as well as their dependencies.

5.2 Future Work

Beyond merely the issues in maintaining existing functionality, discussed in the previous section, there are a number of different directions that this project can go, now that the CPISync algorithm has proven itself efficient.

1) The existing code base is definitely not optimal in its performance. There are many opportunities for improvement, especially in the PalmRPC module, which currently takes as much or more time than the actual CPISync algorithm. One suggested improvement is to have the PalmPilot end the Hotsync session once it has transferred the PC's missing memo texts to the PC, and received its missing memo texts from the PC,

and then incorporate the new memos in the background, or when the user first runs the SyncMemo application after synchronization.

2) The current testing suite is limited by the functionality of the PalmOS v3.3 software for which the Palm-side applications in the SyncMemo project were written. A major simplification to the testing suite would be to have the PalmPilot trigger the test phase from Installation to Synchronization (see Section 2.2.3.4) through the use of a callback DLL. The current mechanism forces the Palm and PC to essentially run separate, non-interacting threads that happen to be synchronized with each other. This modification would add robustness to the testing suite, making it easier to characterize the performance of the entire system.

3) The code for all the built-in Palm functions is freely available (although their PC-side equivalents may be harder to get). It should be possible to extend the SyncMemo code to offer an interface that would fit between an application and its storage on the PalmPilot and maintain the data necessary for the CPISync algorithm. This would require an in-depth knowledge of PalmOS software, as well as a good knowledge of the conduits API.

4) There are new synchronization algorithms besides CPISync and PROBSync that have the potential to perform synchronization with computation linearly proportional to the number of differences. This algorithm is described in the paper by Y. Minsky and A. Trachtenberg entitled: "*Practical Set Reconciliation*" in the 40th Annual Allerton Conference on Communication, Control, and Computing, October 2002. The algorithm would require multiple “rounds” of communication between the PC and PalmPilot (thus, multiple RPCs) but could still use a significant amount of the SyncMemo project’s code base.

5.3 Conclusions

The SyncMemo project was conceived to prove that the CPISync algorithm was implementable and provided superior performance to the wholesale data transfer PDA protocol SlowSync. Not only has the SyncMemo project accomplished this goal, but it has also incorporated an alternative synchronization mechanism known as PROBSync that can improve the performance of synchronizing small data sets while maintaining the ability to synchronize large ones. Furthermore, the SyncMemo project has provided valuable experience in constructing a synchronization framework between the PC and a Palm PDA. There is plenty of more work to do on the code base itself: debugging and tweaking performance, but the primary purpose of the SyncMemo project has been met, and CPISync and the related PROBSync algorithms have been shown to provide real advantages over existing methods of synchronization.